苏 州 大 学 实 验 报 告

院、系	计算机学院	年级专业	23 软件工程	姓名	朱金涛	学号	2327406014
课程名称	操作系统实践					成绩	
指导教师	王红玲	同组实验者	无		实验日期	2025 年 10 月 17 日	

实 验 名 称　添加 Linux 系统调用

一、 实验平台和实验时间

实验平台：带有 Linux 源代码的 Linux 平台。
实验时间：4 个小时

二、 实验目的

在理解系统调用基本原理的基础上，通过修改 Linux 内核相关内容，新增一个系统调用，由此掌握 Linux 系统下系统调用的实现和调用过程。

三、 实验内容

为 Linux 新增一个系统调用，重新编译内核并安装，并编写 C 语言程序调用该系统调用。要求：使用教材上提供的内核编译法完成。如始终未能成功编译内核，也可采用内核模块法完成。

具体要求：①完成一个简单系统调用（无参数）。②完成一个带 2 个参数的系统调用函数，要求输入 int a，b，输出为 a 与 b 的和。

四、 实验步骤和结果

（一）实验步骤

1. 进入内核源码目录并获得 root 权限

使用命令：

```
sudo -i
```

切换到 root 用户，确保具有编译内核的权限。

2. 添加第一个系统调用（无参数的 helloworld）

（1）在 arch/x86/entry/syscalls/syscall_64.tbl 文件末尾添加系统调用号（我的内核中最后一条系统调用号为 547）：

```
546     x32     preadv2                 compat_sys_preadv64v2
547     x32     pwritev2                compat_sys_pwritev64v2
548     64      helloworld              sys_helloworld
# This is the end of the legacy x32 range.  Numbers 548 and above are
# not special and are not to be used for x32-specific syscalls.
```

（2）在 kernel/sys.c 文件中添加函数实现：

```
SYSCALL_DEFINE0(helloworld){
        printk("helloworld!");
        return 1;
}
```

（3）保存后重新编译并安装内核：

```
make -j16
make modules_install
make install
reboot
```

3. 验证第一个系统调用

编写 test_hello.c：

```
#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>
int main() {
    long res = syscall(548);
    printf("sys_helloworld returned: %ld\n", res);
    return 0;
}
```

编译运行：

```
tt@tt-VMware-Virtual-Platform:~/文档$ gcc hello.c
tt@tt-VMware-Virtual-Platform:~/文档$ ./a.out
System call sys_helloworld reutrn 1
```

使用 dmesg | tail 查看内核日志：

```
tt@tt-VMware-Virtual-Platform:~/文档$ sudo dmesg | tail
[sudo] tt 的密码：
[   37.049546] Code: Unable to access opcode bytes at 0x7b49838f7263.
[   37.049549] RSP: 002b:00007b498256adc8 EFLAGS: 00000246 ORIG_RAX: 00000000000
000ca
[   37.049557] RAX: ffffffffffffe00 RBX: 00005b6f74853630 RCX: 00007b49838f728d
[   37.049561] RDX: 0000000000000005 RSI: 0000000000000080 RDI: 00005b6f74853640
[   37.049564] RBP: 00007b498256ae00 R08: 0000000000000007 R09: 00005b6f7485dff0
[   37.049568] R10: 0000000000000000 R11: 0000000000000246 R12: 00007b498256b648
[   37.049572] R13: 0000000000000005 R14: 00005b6f74853640 R15: 0000000000000002
[   37.049577]  </TASK>
[   37.049581] ---[ end trace 0000000000000000 ]---
[   72.632545] helloworld!
```

说明系统调用成功执行。

4. 添加第二个函数调用（带两个参数的 add）

（1）在 syscall_64.tbl 文件中继续添加新行：

```
548     64      helloworld              sys_helloworld
549     64      add                     sys_add
```

（2）在 kernel/sys.c 文件末尾添加函数实现：

```
SYSCALL_DEFINE2(add, int, a, int, b)
{
    return a + b;
}
```

（3）保存并重新编译内核：

```
make -j16
make modules_install
make install
reboot
```

5. 验证第二个系统调用

编写 test_add.c：

```
#include <stdio.h>
#include <sys/syscall.h>
#include <unistd.h>
int main() {
    int a = 10, b = 20;
    long res = syscall(549, a, b);
    printf("sys_add(%d, %d) = %ld\n", a, b, res);
    return 0;
}
```

运行结果：

```
tt@tt-VMware-Virtual-Platform:~/文档$ gcc test_add.c -o test_add
tt@tt-VMware-Virtual-Platform:~/文档$ ./test_add
sys_add(10, 20) = 30
```

结果正确，系统成功调用！

（二）实验结果

1. 成功在内核中添加了两个新的系统调用：

- 548 号：helloworld——输出字符串到内核日志。
- 549 号：add——实现两个整数的加法运算并返回结果。

2. 用户态程序均能正确调用相应系统调用。

3. 通过 dmesg 查看系统日志，能看到内核输出的 printk 内容，说明系统调用已正确注册并执行。

五、 实验总结

问题解决：一开始按照 PPT 中的教程改写 sys.c 文件，采用：

```
asmlinkage long sys_helloworld(void){
        printk("helloworld!");
        return 1;
}
#endif /* CONFIG_COMPAT */
-- 插入 --
```

发现编译后报错，问题出现在链接阶段，具体是连接器（ld）无法找到__x64_sys_helloworld 的定义。查询资料后得知在最新的 6.17 内核中，传统的函数定义方法并不完全适用。需要以宏定义的形式才能被系统识别：

```
SYSCALL_DEFINE0(helloworld){
        printk("helloworld!");
        return 1;
}
```

实验小结：

本次实验通过修改 Linux 内核源码，成功实现了系统调用机制的扩展。实验过程中，首先理解了系统调用在用户态与内核态之间的桥梁作用，然后分别完成了两个系统调用的添加与验证：一个无参数的 helloworld 调用，用于在内核日志中输出字符串；另一个带两个整型参数的 add 调用，实现了两个数的加法运算并返回结果。在操作中经历了编译报错、符号未定义等问题，经过查阅资料后了解到新版本内核需使用 SYSCALL_DEFINE 宏定义才能正确生成系统调用入口。通过本次实验，我深入理解了系统调用的实现原理、内核编译流程及内核与用户空间的交互机制，增强了对操作系统底层结构的认识，并提升了实际调试与问题分析能力。